Atlan - Backend Task

Deepankar Arun Jain

Problem statement

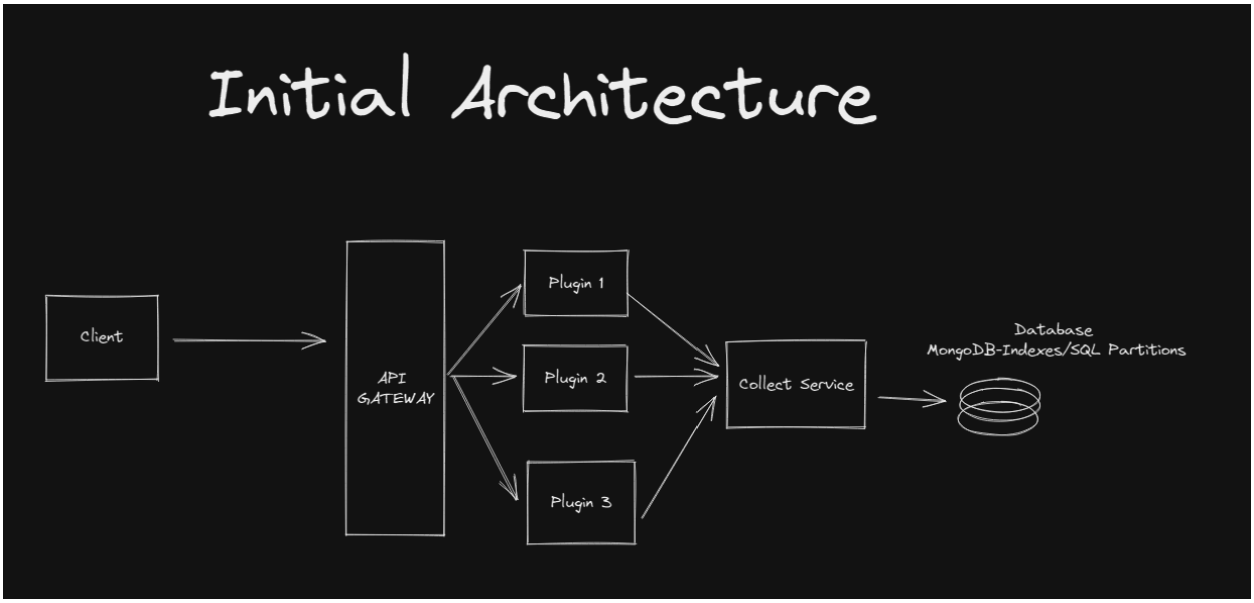
Design a sample schematic for how you would store forms (with questions) and responses (with answers) in the Collect data store. Forms, Questions, Responses and Answers each will have relevant metadata. Design and implement a solution for the Google Sheets use case and choose any one of the others to keep in mind how we want to solve this problem in a plug-n-play fashion. Make fair assumptions wherever necessary. Solve this problem in such a way that **each new use case can just be “plugged in”** and does not need an overhaul on the backend. Imagine this as a whole ecosystem for integrations. We want to optimize for **latency and having a unified interface** acting as a middleman**.**

Eventual consistency is what the clients expect as an outcome of this feature, making sure no responses get missed in the journey. Do keep in mind that this solution **must be failsafe**, should eventually recover from circumstances like power/internet/service outages, and should **scale to cases like millions of responses** across hundreds of forms for an organization.

There are points for details on how would you benchmark, set up logs, monitor for system health, and alerts for when the system health is affected for both the cloud as well as bare-metal. **Read up on if there are limitations on the third party** (Google sheets in this case) too, a good solution keeps in mind that too.

Initial Architectures

The problem statement, involved emphasizing the fact of using a “pluggin in” or plug and play architecture for the services. Considering the need to make it plug and play, The first ideology towards the problem statement was considering an open ended solution for maintaining plugins, for instance, validation of savings and monthly income, appending data to google sheet before hiting the main backend service. The architecture like the below:



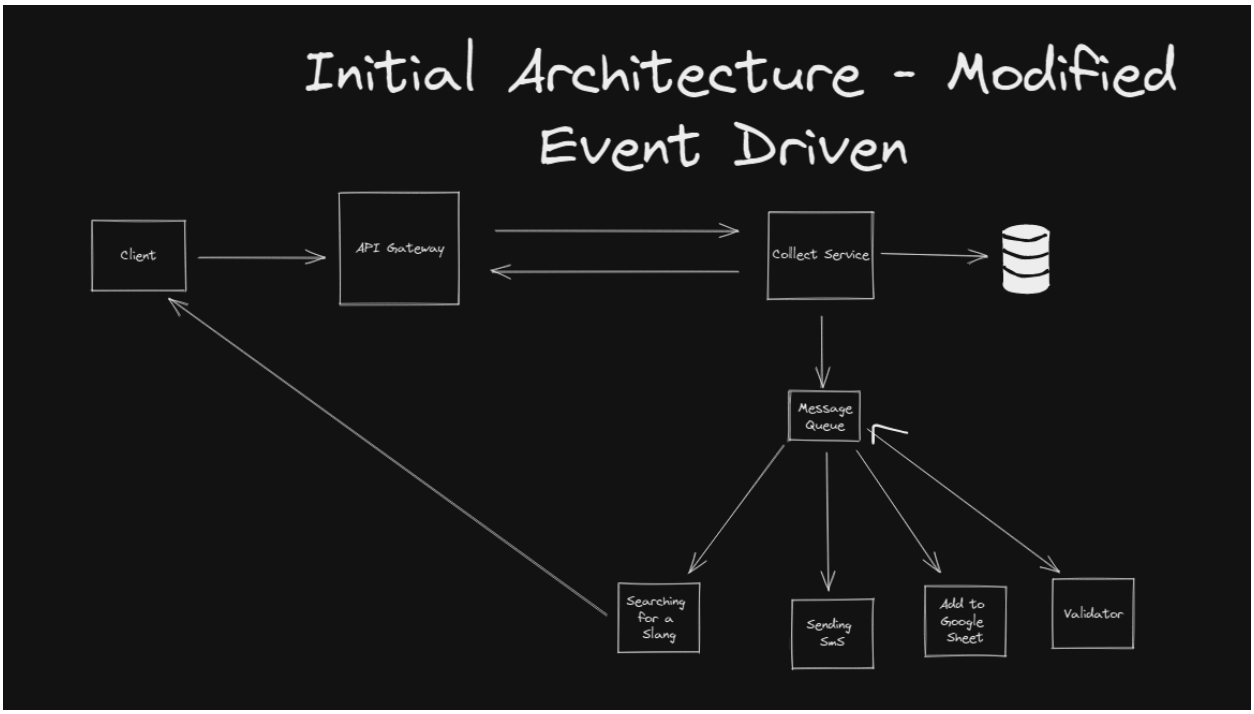
With respect to the benefits of the architecture, Considering the need for scalability and consistency, this architecture helps to follow up and operate on both the factors. The different plugins can be horizontally or vertically scaled, working towards a consistent solution can be achieved by proper routing and keeping interfaces with proper designs and test cases.

The shortcomings of the architecture exist by actually being implemented in a plug and play architecture asked in problem statement. Considering a happy case, where the client wishes to have a single plugin, then the plugin works well, with a single API call works towards the collect service writing to the database (if any). But, Imagine the case, where the client wishes to have two plugins, Now, According to the above architecture, It needs to make two API Calls to the collect service, proning to duplicate entries and more number of API Calls for the architecture. This is where we fail in first place.

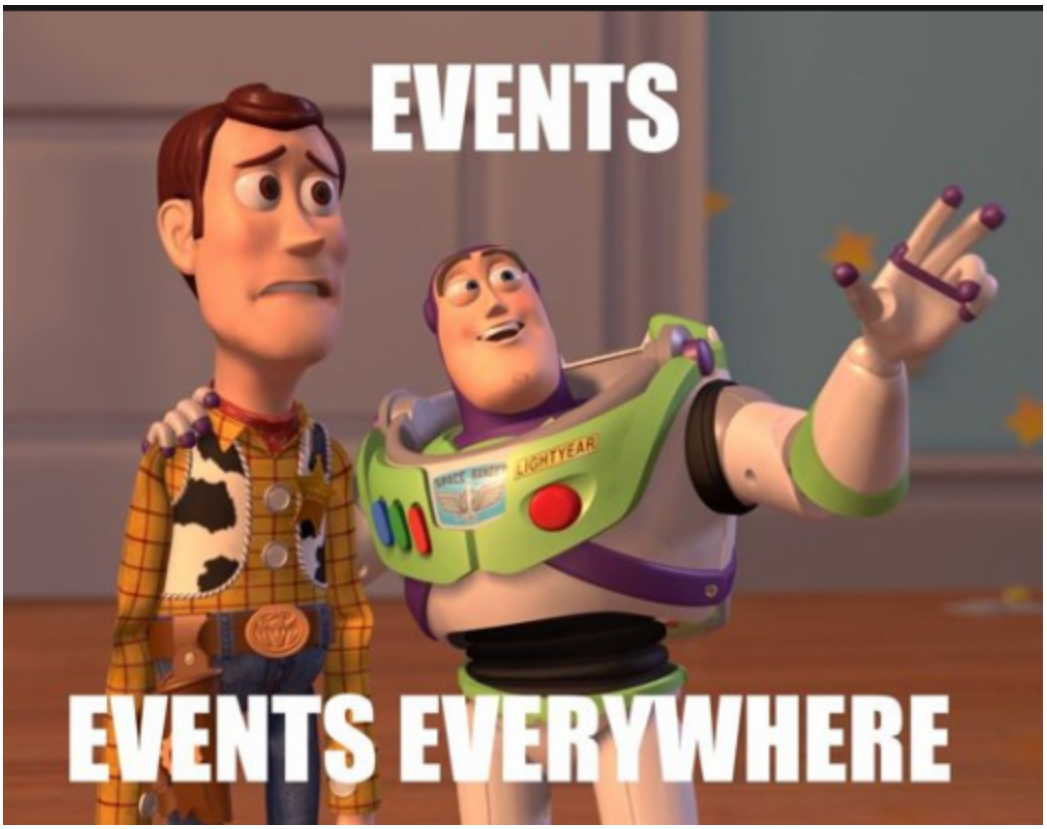
The System also fails in the situation to reach out a proper response in place the collect service responds properly, but the internal error in plugin is developed. Imagine the service to append to the spreadsheet once the data is inserted, Running the same as plugin, A structured approach is to wait for the collect api and upon successful completion, append the data to the spreadsheet, But the happy case misses when we appending the data service fails, and henceforth It'll be difficult extracting the session for which we missed the data insertions. With the emphasis on plug and play architecture. It needs utilising an event-driven approach. The current approach will want us to configure the api gateway by creating a shared layer of middleware plugins. Eventually, the plug and play will involve a comprehensive structure to optimise the operations of adding and removing services.

Considering the same, I modified the approach to make it event driven completely following a plug and play featured.

Modifying Initial Architectures to an Event Driven Architecture



The above architecture is an event driven architecture where each service can be added (plugged in) or removed (plugged out) from the system upon the introduction of any event. Considering an example for the same, An event can be recognised as inserting a data in the database, which is created from the collect service. Upon insertion of the data, the event is passed on the subsequent services as a publisher-subscriber model and the services can avail their task such as sending an sms or adding the data to a spreadsheet.



With the advantages of the architecture involve adding any number of services to the system without any external configuration. This helps us in also providing a more scalable and a consistent network of working for our system.

The above architecture is fully event driven and now becomes the source of its own problem. Noticing properly that, every service is working independently and hence making it difficult to respond. Considering the plugins to search for a slang in data. This service needs to respond to the client for the result. Imagining the request creating an event and the queue has a lot of events inside it, then subscribing to the events will take some ‘x’ amount of time. After this ‘x’ amount of time when the service would’ve responded to the request, It needs to show back the result to the client. Eventually, We can’t block the client until the process is completed (ie been fetched and computed) and hence we need sockets to come into play to respond to the client. So the end problem becomes more tedious with working with sockets and ensuring that client receives the response very soon with any delay. Eventually the need to notice is that, searching for a slang in data creates a dependent event for which the response is

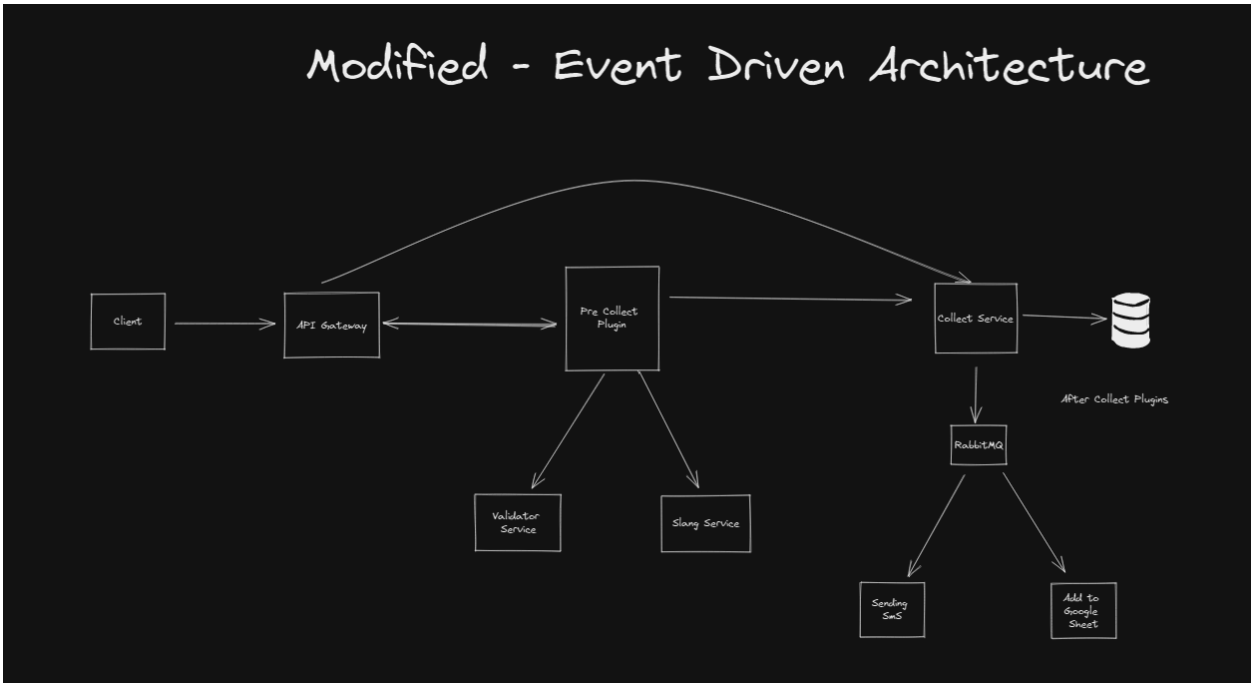
dependent on the service unlike sending sms which is independent for the client to know about.

Another case of failure is noticing the validation service. The system flow is communicated by the collect service inserting the data first and then pushing to the message queue. If the validation service is put to action, and it does flag off a data entry, this means that the same entry needs to be deleted from the database making the transactions and database call heavy. This is also followed by a delay in the response to the client about a flagged off entry.

By two failed architectures, We realised that some requests/events require to communicate to the client and hence are cilent dependent and others such as

SMS/Spreadsheet can be client independent and can be achieved from event-driven nature.

Modifying the Event Driven Architecture

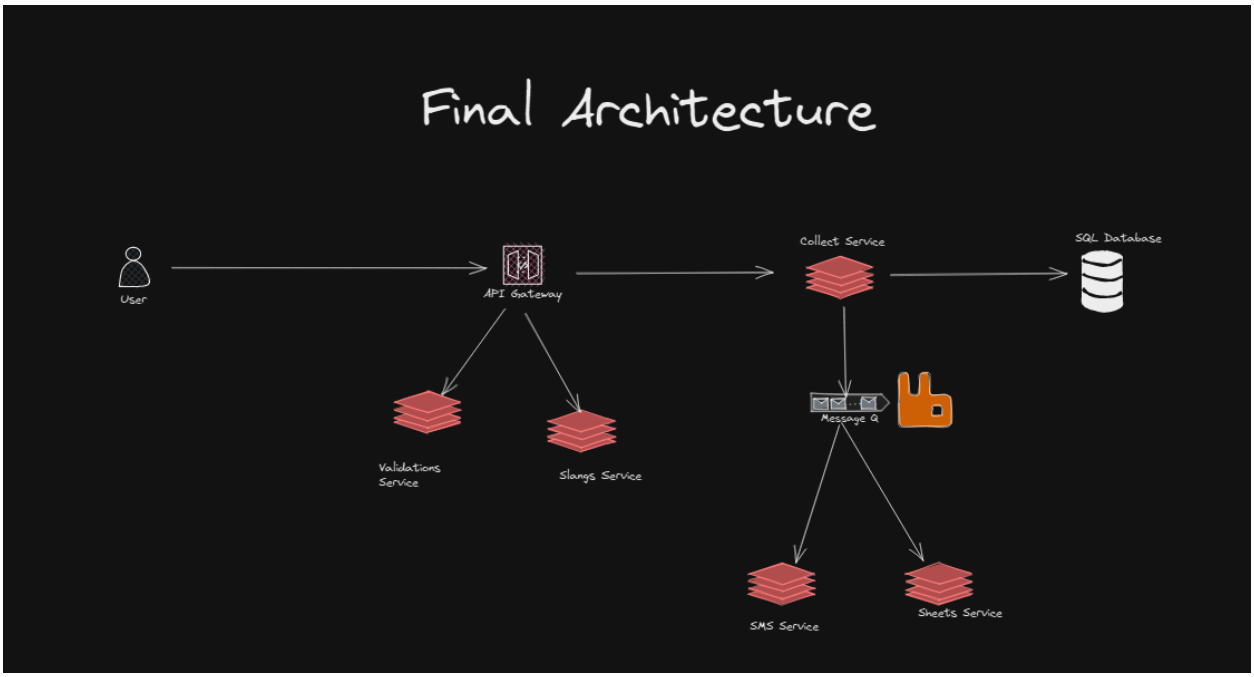


The above modified architecture works by segregating the services based on being dependent of the client and independent of the client. The ones which are client dependent are termed as Pre-Collect Plugin and the independent ones are termed as After Collect Plugins. As discussed above, validation service and slang service can be served as Pre-Collect Plugin and SMS Service and Google Sheets insertion are After-Collect Plugin.

The client makes request to the API Gateway, which depending on the identification as Pre-Collect Service or Post-Collect Service routes to the service. The Pre-Collect Service responds to the client on serving the request, while Post-Collect Service only issues a message to the client if the service has acknowledged the request.

Extending the advantages of the event driven architecture, We've managed to successfully make a scalable and a consistent system, with segregation and decision to make a plugin be added before the collect service or after the collect service. Eventually, the thing to realise is we've made a lot decoupling. The Pre-Collect Service plugins can be further integrated to api gateway to directly route the request, thus ending the extended route mechanism which the system is working at currently.

Final Architecture Proposed

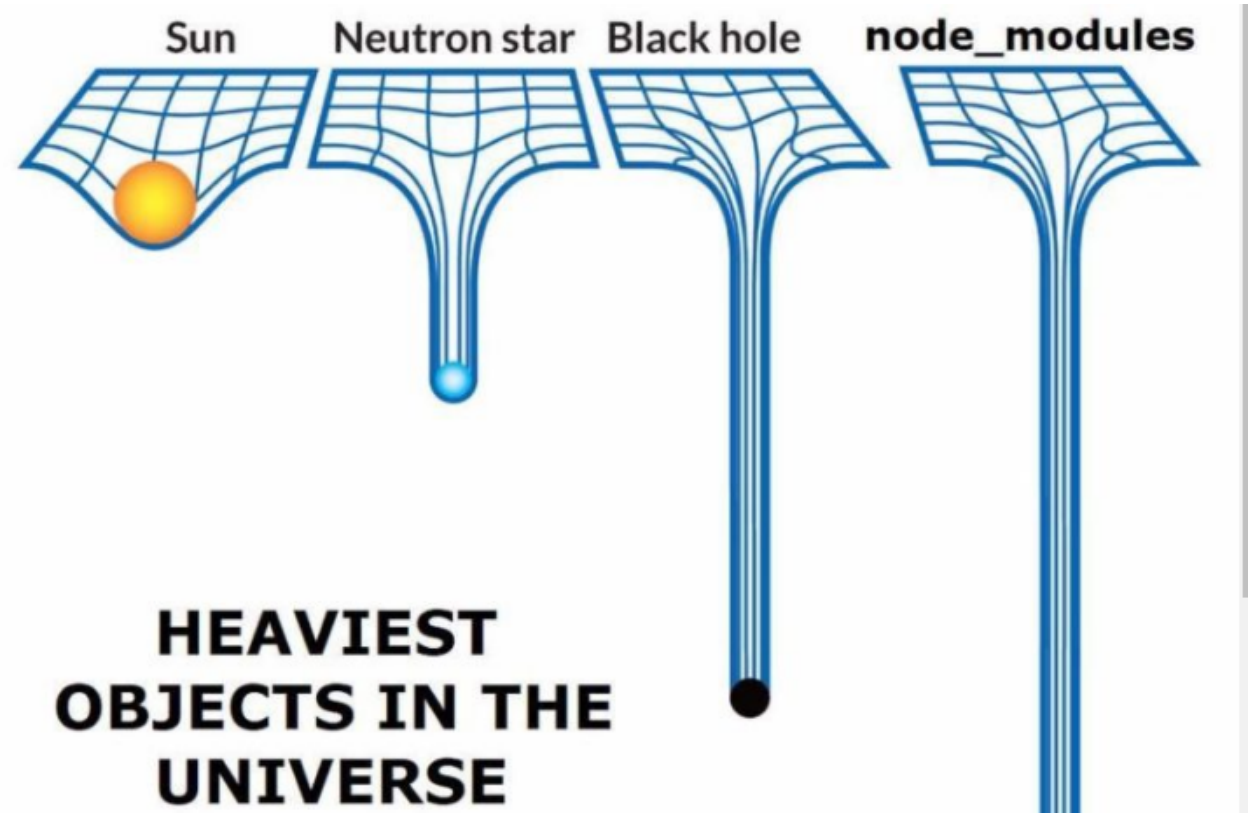


The final architecture incorporated the benefits of event-driven design as well as the need to create segregated microservice for client dependent services. Henceforth, Our system can be scaled for individual services and corporate more consistent behaviour in terms of solution to add numerous service to the system. Though configuration have to be made in API Gateway for pre-collect service(example: Validate Service and Slangs Service). Since Message Queue stores the event till they're processed, any failure in Post-Collect Service can be maintained refreshed and the events become consistent at front. The API Gateway can be monitored with Application Monitoring Tools such as New-Relic and Grafana and logs to keep updated about the status of the request and error rate.

Understanding the Tech Stack

- Postgresql for Database
    - Considering, the data structure and schema to be same, It’s very well to consider SQL Database. SQL databases are efficient at processing queries and joining data across tables, making it easier to perform complex queries against structured data, including ad hoc requests.
- Kong for API Gateway
    - Recognised as the most popular open source API Gateway with necessary tech support. I also tried using Apisix by Apache, but couldn’t configure it and thus didn’t worked
- RabbitMQ for Message Queue
    - Popular open source message queue. This was my first time working with message queues in general, so I took RabbitMQ.

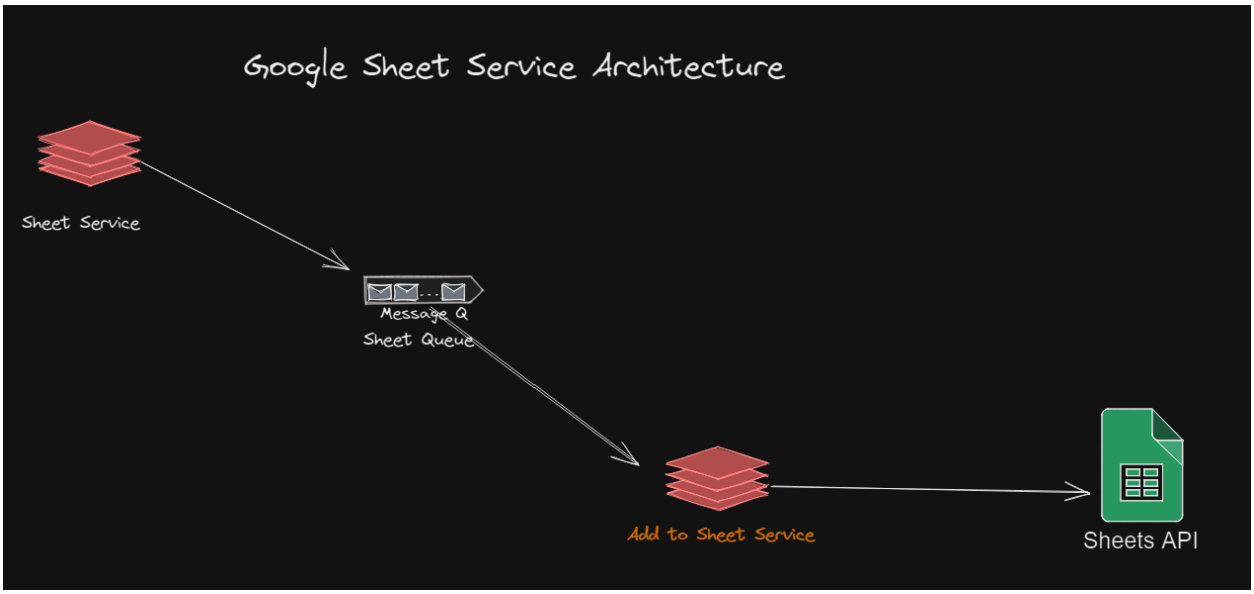
- NodeJS along with Typescript: All the microservices are developed on NodeJS with Typescript
    - I had previous experience developing with NodeJS and Typescript for a clean architecture
- Docker
    - To containerise the application and using Kong service



Understanding Google Sheets Service and Validation Service

The two service that I chose to develop was Google Sheet Service and Validation Service. While Google Sheet Service is independent of the client, Validation Service is dependent on the client and hence has been routed via API Gateway ie Kong.

Google Sheets Service Architecture



With respect to the external Google Sheets API, The limitation exists as follows :

Quotas		
Read requests	Per day per project	Unlimited
	Per minute per project	300
	Per minute per user per project	60
Write requests	Per day per project	Unlimited
	Per minute per project	300
	Per minute per user per project	60

Therefore, If we would’ve connected the Sheets API directly to Sheet Service, then a case would’ve been made where more than 300 read/write request could’ve been made since Sheet Service looks up for the events from Collect Queue.

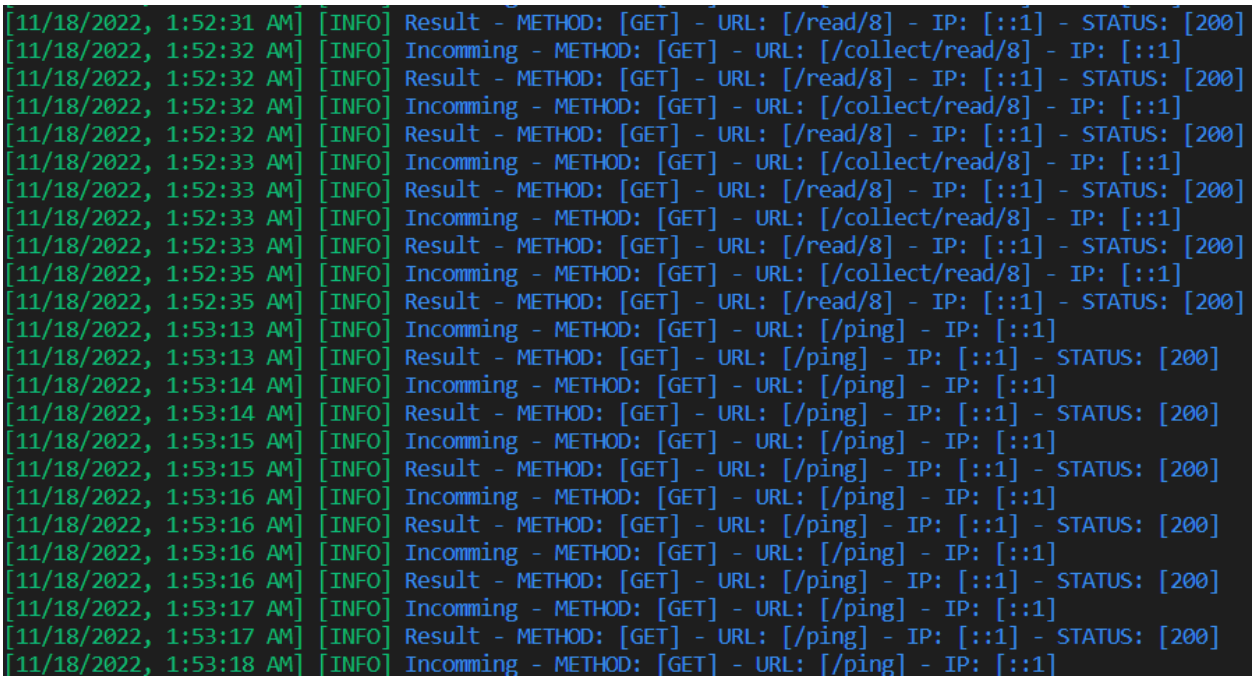
Eventually, To solve the issue, the solution was to use a delay queue which delays the messages to be consumed by subscriber by a certain milliseconds. For instance, A delay of 500 milliseconds will delay the message to be consumed by a certain service

by 500 milliseconds. Hence, We can avoid a lot of request being embedded to Sheets API. For this, I used delayed queue plugin in RabbitMQ.

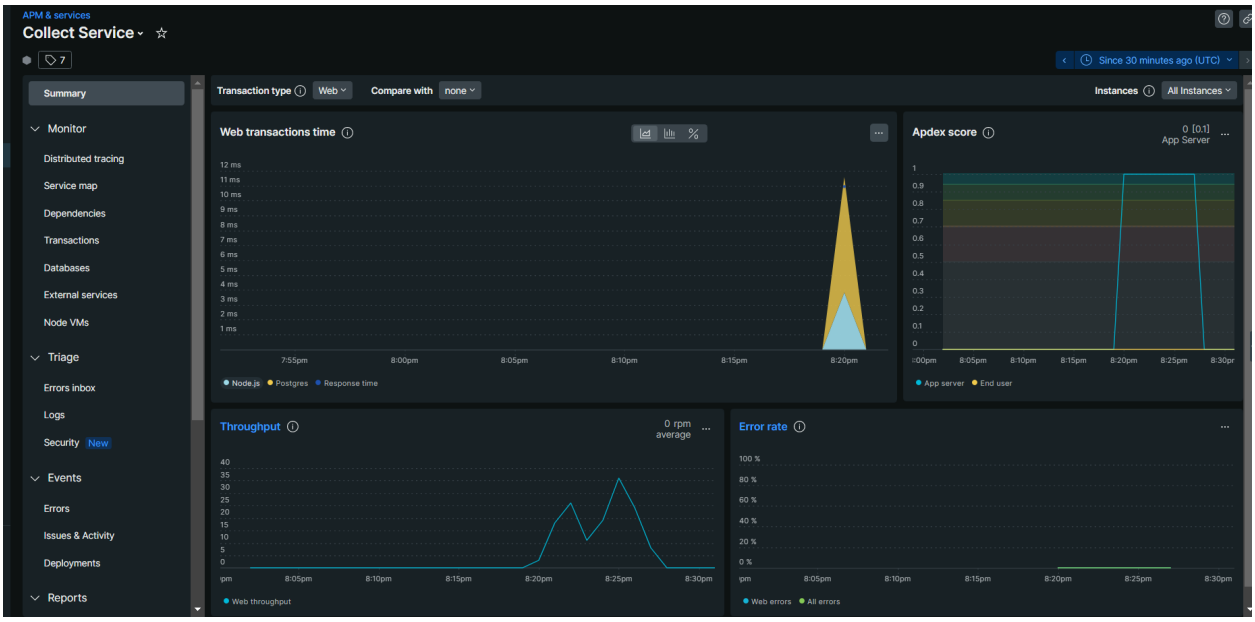
Summarising the above architecture, The Sheet Service has subscribed from Collect Service for insert events in the database. As soon as the event is published, Sheet Service then pass it to another delayed queue, which is further subscribed by add to sheet service which appends the data from Sheets API.

Monitoring and Logging

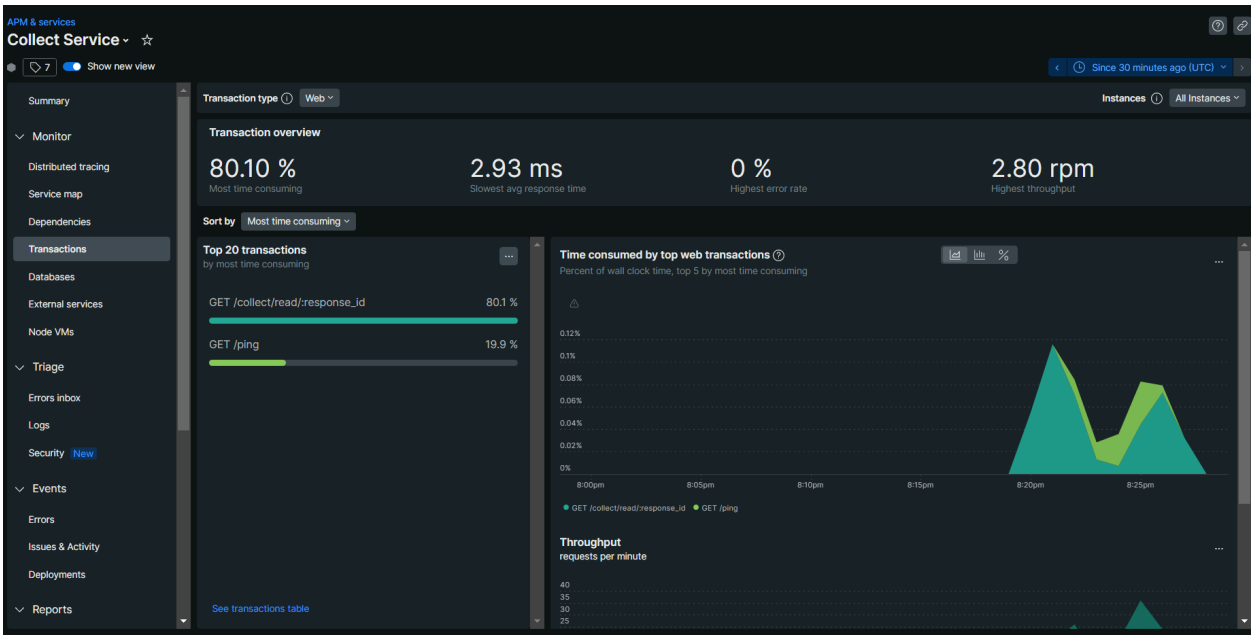
For the purpose of monitoring,, I've integrated New Relic Application Monitoring Manager and for the purpose of logging, The Logging must be typically done via Winston Logger(For simplicity, I've used local logging of the messages using chalk and local logs in Node).

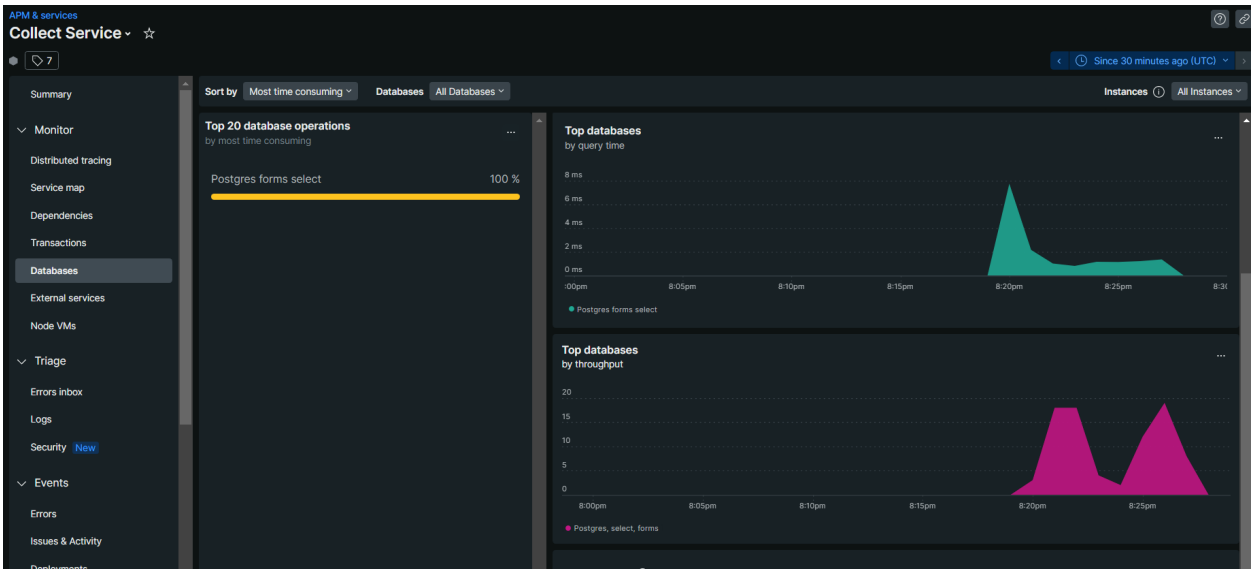


The above log is using the chalk package and console.log. This log can be outputted to a file to perform analysis.

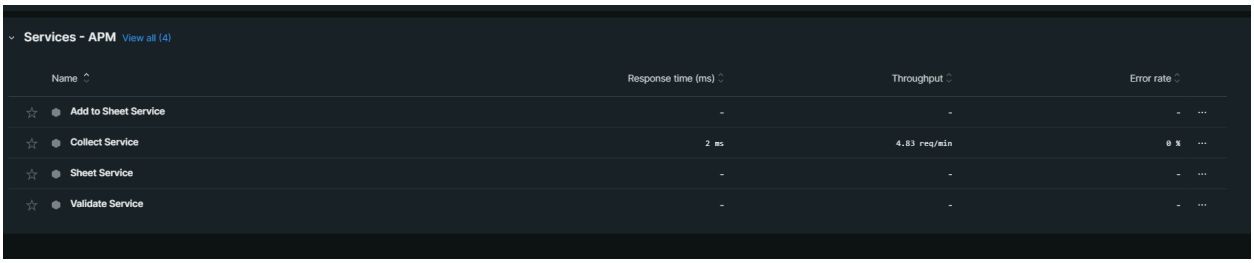


The above dashboard is of new-relic application monitoring tool. It provides a comprehensive view for different route request and error rate, with plugins to monitor and generate alerts based on failure.





These are some of the features of the application monitoring system, and can be configured for each service as follows:

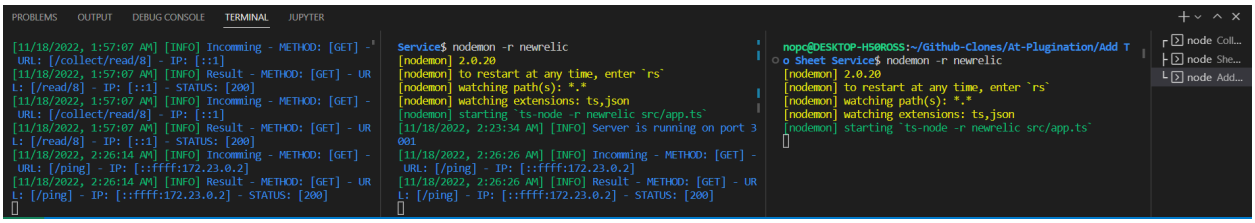


Configuration of API Gateway

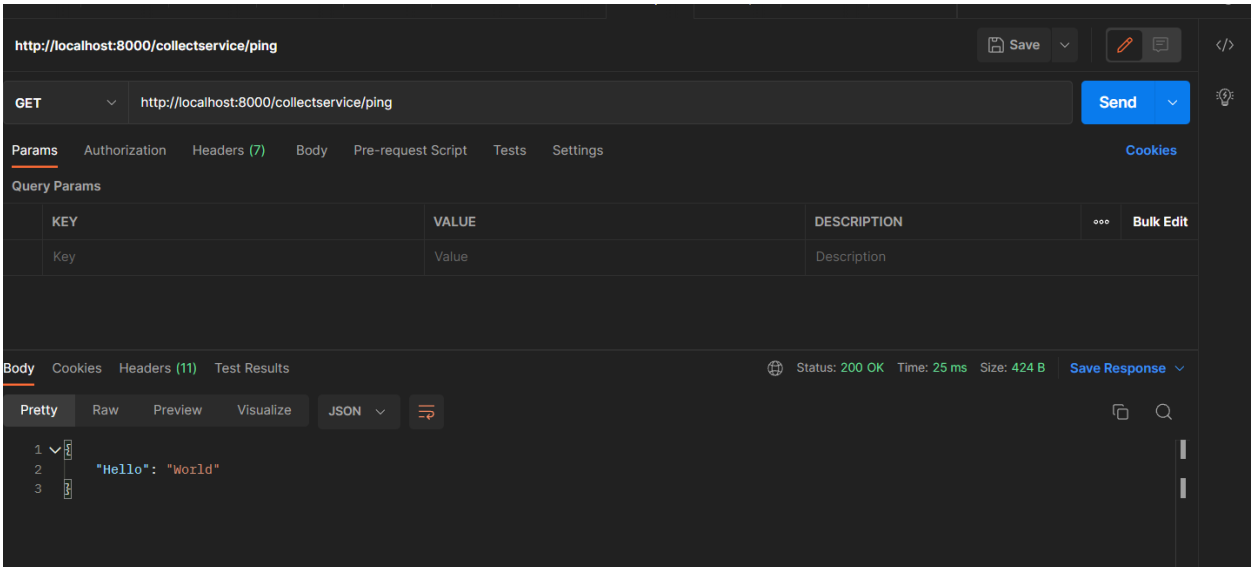
As mentioned in the architecture diagram, The API Gateway is responsible for routing the request to collect service and validation service for being independent and dependent services to client.

```
kong.yml > [ ] services > {} 1 > [ ] routes > {} 0 > [ ] paths > abc 0
1   _format_version: "2.1"
2
3   services:
4     - name: validation-service
5       url: http://172.17.0.1:3003
6       routes:
7         - name: validation-route
8           paths:
9             - /validation
10    - name: collect-service
11      url: http://172.17.0.1:3000
12      routes:
13        - name: collect-service-route
14          paths:
15            - /collectservice
```

The services running on the ports as shown as below:

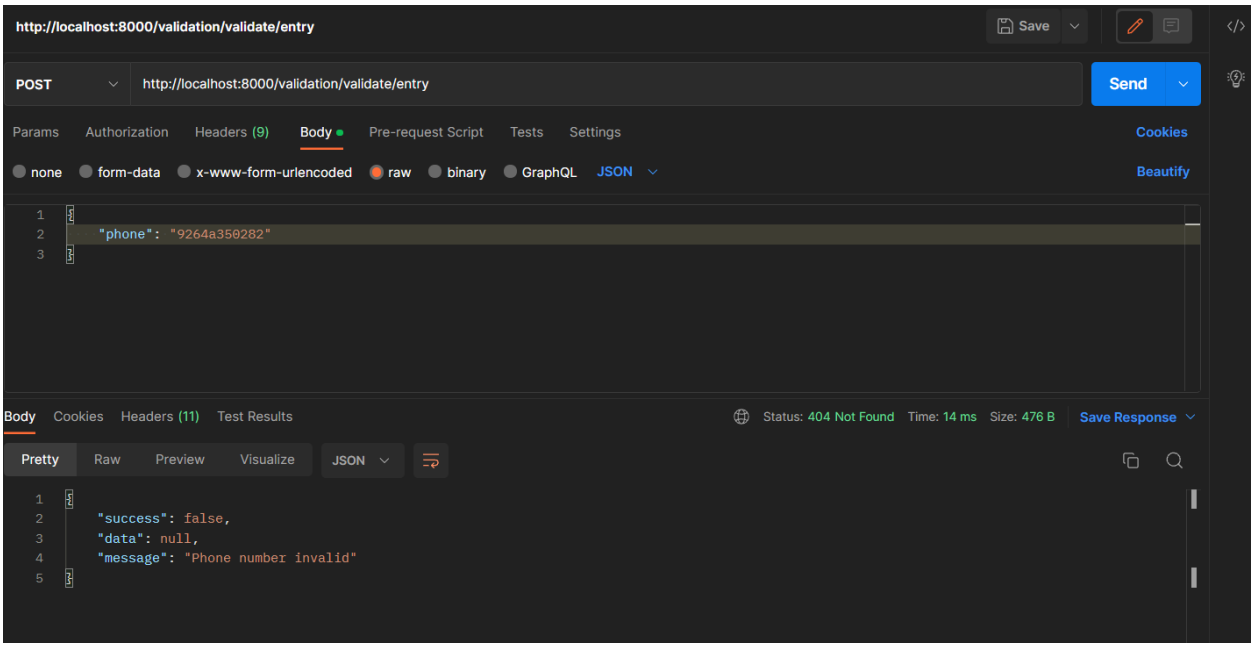


Checking if the route is working or not

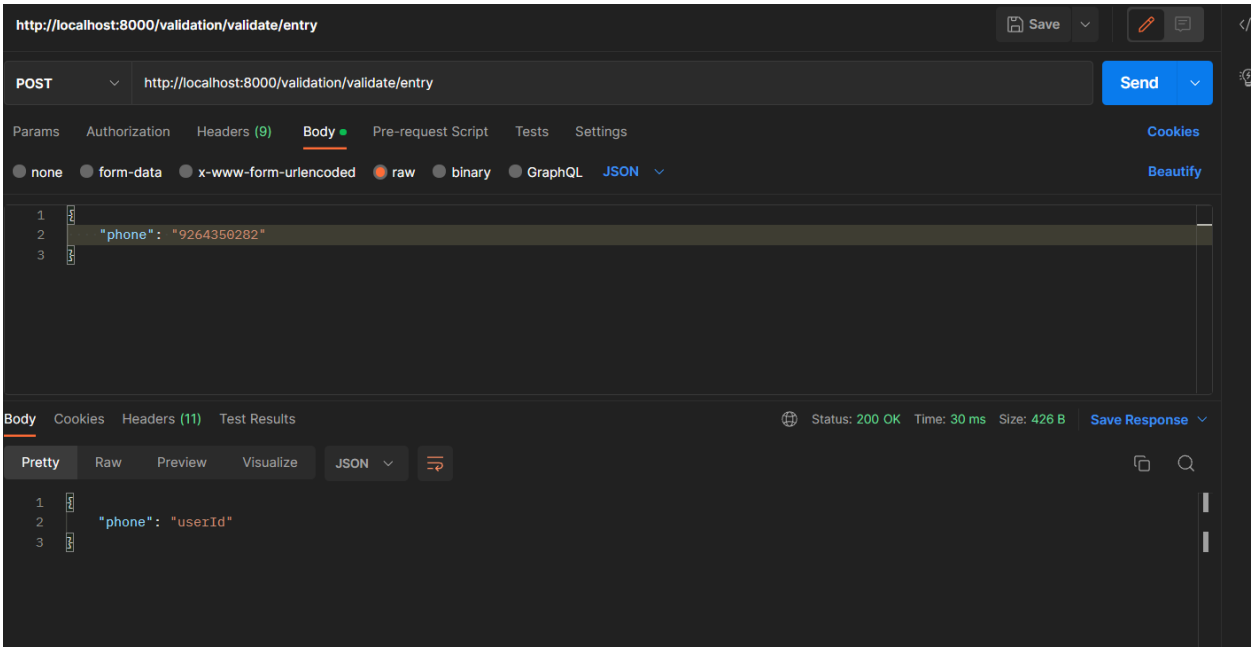


Validation Service

The validation service in this case, checks if a phone number is valid or not. If the number is not valid, It generates a message denoting the same.



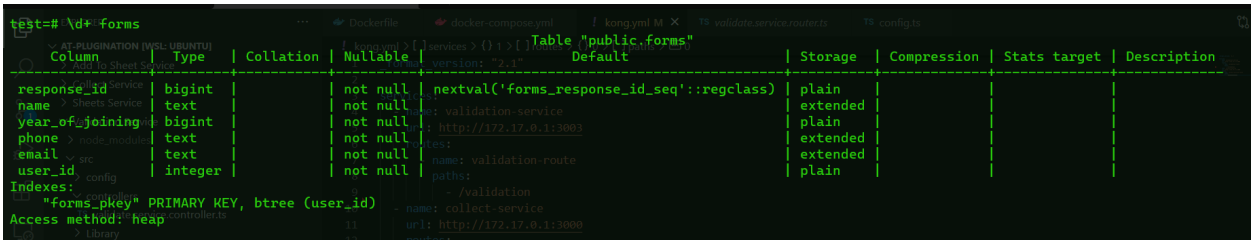
If the phone number is valid, It returns a status of 200 with a different response (Since, it’s just a mock of the actual case)



Collect Service

The collect service is responsible for storing the data into the database. The Schema for the database is as follows :

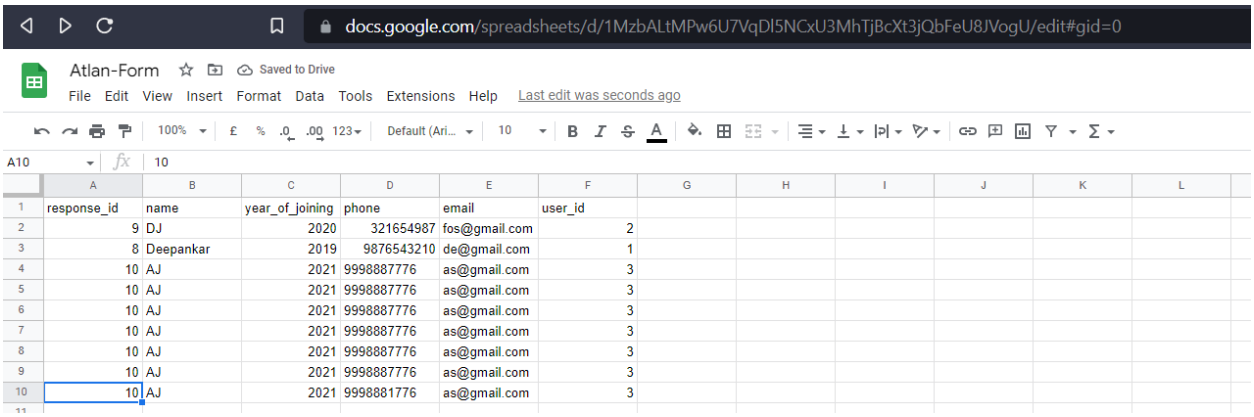
Note: This is a mock database and therefore the fields are very few and just for understanding purpose



As it can be seen, I’ve made user_id as a primary key reason being that a particular user shouldn’t trying posting the same form again.

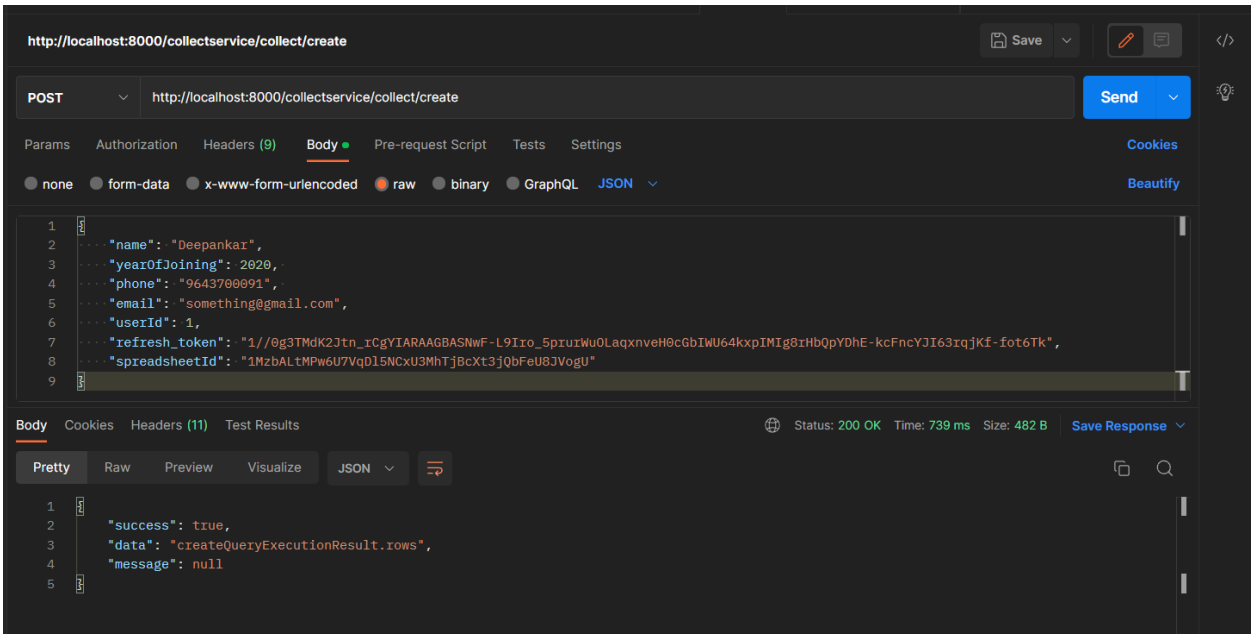
Google Sheets Link for accessing the sheets. The building logic involves to make an entry into the google sheets as and when the data is inserted into database. For

Illustration purpose, I've commented the code for inserting the data in database for simplicity. The Google Sheets looks like this as of now:

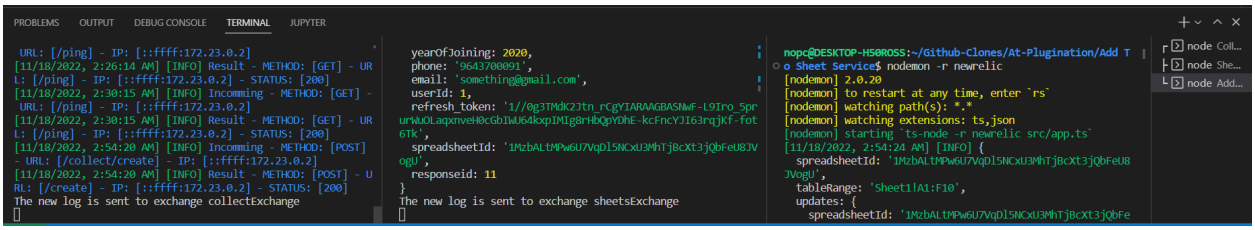


	A	B	C	D	E	F
1	response_id	name	year_of_joining	phone	email	user_id
2	9	DJ	2020	321654987	fos@gmail.com	2
3	8	Deepankar	2019	9876543210	de@gmail.com	1
4	10	AJ	2021	9998887776	as@gmail.com	3
5	10	AJ	2021	9998887776	as@gmail.com	3
6	10	AJ	2021	9998887776	as@gmail.com	3
7	10	AJ	2021	9998887776	as@gmail.com	3
8	10	AJ	2021	9998887776	as@gmail.com	3
9	10	AJ	2021	9998887776	as@gmail.com	3
10	10	AJ	2021	9998881776	as@gmail.com	3

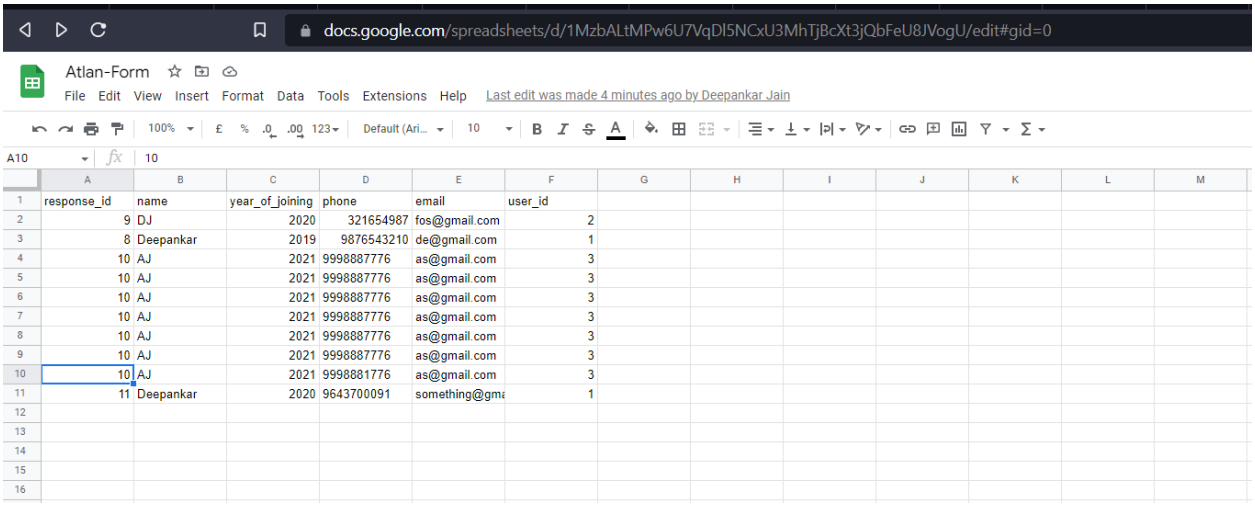
Now Let us try to make an API Call to the collect service and checks for the logs on all the service as well Google Sheet



As seen above, the request contains the basic details, along with spreadsheetid and refresh_token. For Generating the same, I've already added the auth code in the sheet service.



As seen from the above log, The Collect service emits an event to the collectExchange queue and sheet service which is in the middle terminal, sends the same request to sheetsExchange which is fetched by “Add to Sheet” Service and appends the data to the Google Sheets.

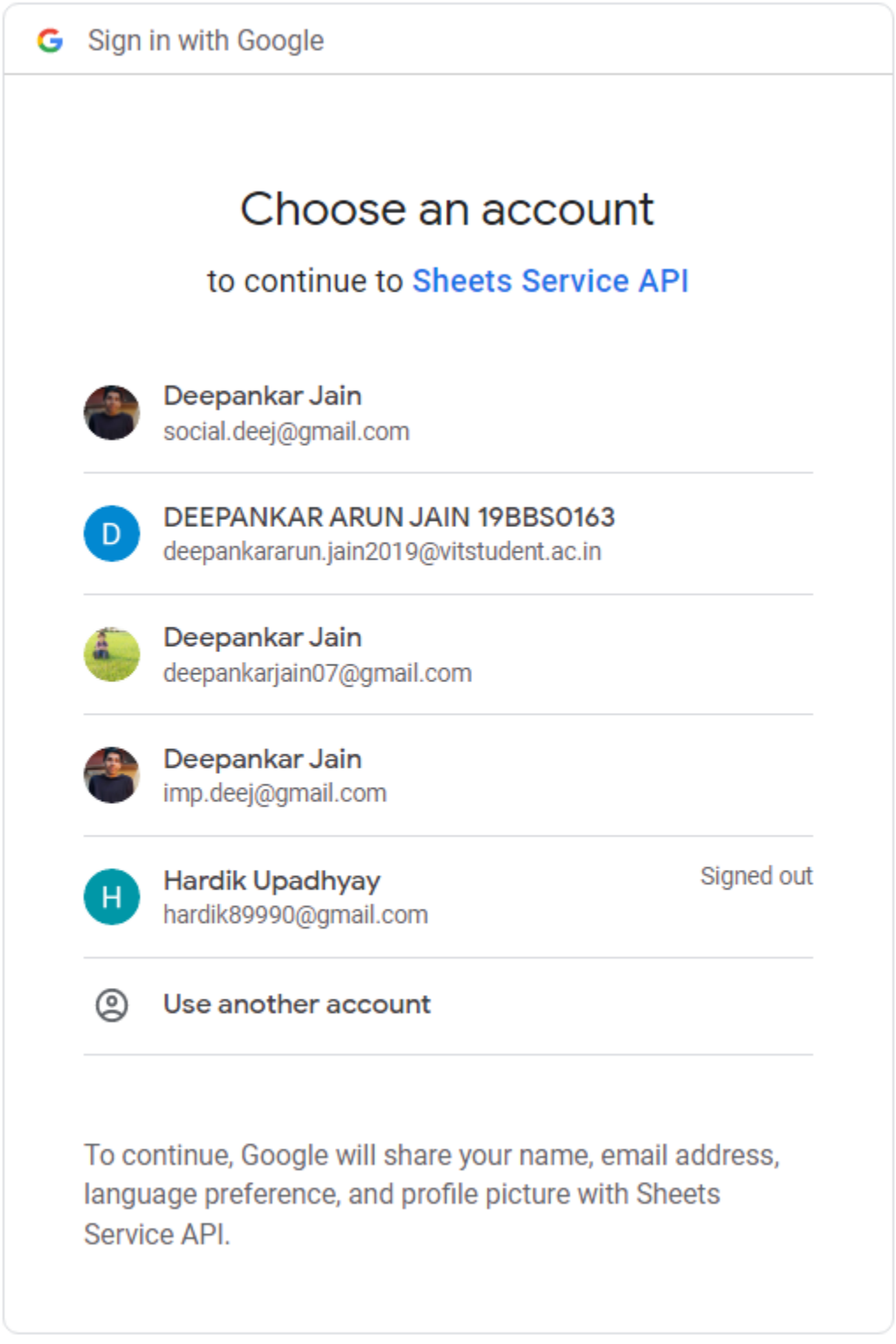


	response_id	name	year_of_joining	phone	email	user_id
2	9	DJ	2020	321654987	fos@gmail.com	2
3	8	Deepankar	2019	9876543210	de@gmail.com	1
4	10	AJ	2021	9998887776	as@gmail.com	3
5	10	AJ	2021	9998887776	as@gmail.com	3
6	10	AJ	2021	9998887776	as@gmail.com	3
7	10	AJ	2021	9998887776	as@gmail.com	3
8	10	AJ	2021	9998887776	as@gmail.com	3
9	10	AJ	2021	9998887776	as@gmail.com	3
10	10	AJ	2021	9998881776	as@gmail.com	3
11	11	Deepankar	2020	9643700091	something@gma	1

As it can be seen from the above form, that the data has been added to the Google Sheets.

Google Sheet Auth Service (Optional)

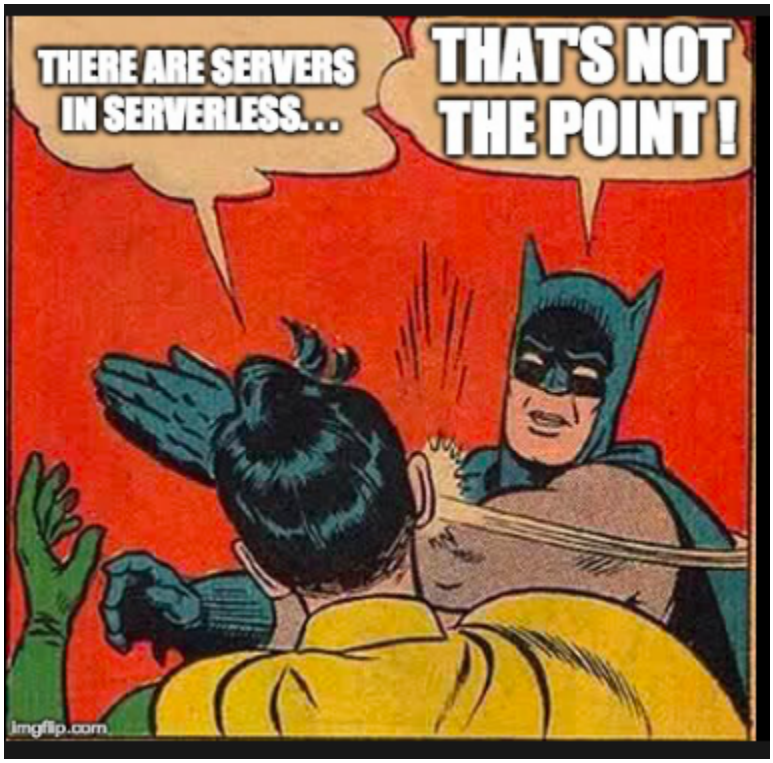
As it was seen in the Collect Service API, We required refresh_token as well as spreadsheetId. To generate the same, You need to log on to localhost:3001/auth to get the below interface.


Sign in with Google
Choose an account
to continue to Sheets Service API
Deepankar Jain
social.deej@gmail.com
D
DEEPANKAR ARUN JAIN 19BBS0163
deepankararun.jain2019@vitstudent.ac.in
Deepankar Jain
deepankarjain07@gmail.com
Deepankar Jain
imp.deej@gmail.com
H
Hardik Upadhyay
hardik89990@gmail.com
Signed out
Use another account
To continue, Google will share your name, email address, language preference, and profile picture with Sheets Service API.

As the API Key is generated for test development, Only selected accounts are added to login and get the key. The below is the final response generated. As it can be seen that refresh_token is available in the response and can be used from here. The next step involves accessing Google Sheet API to get the spreadsheedId to make changes.

Improvements

- The function for add to sheet service can be made serverless, amounting the same for the fact that It needs to responds to an event. Therefore, The same function could’ve been interesting if perfomed via AWS lambda, adding more scalability and reliability to the system.



- The current implementation is a mock representation of the real world analysis, and therefore has many shortcomings from code point of view. Inevitably, There’s more to add into the architecture such as log saving and transactions to recover back from failure and power outage.

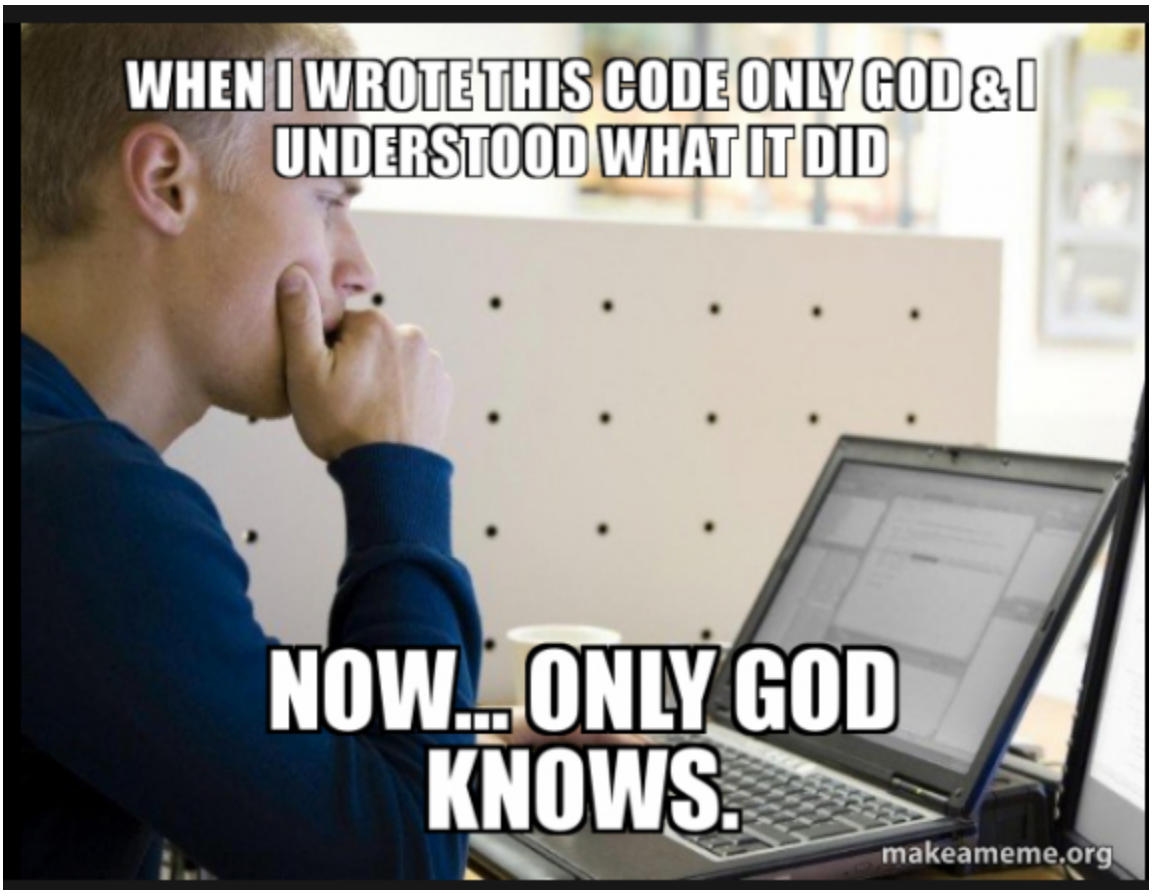
Reviewing Code

- Google OAuth took a day to finish up. It was a bit difficult to test and make changes everytime while following the documentation integrating both OAuth and Google Sheets API.

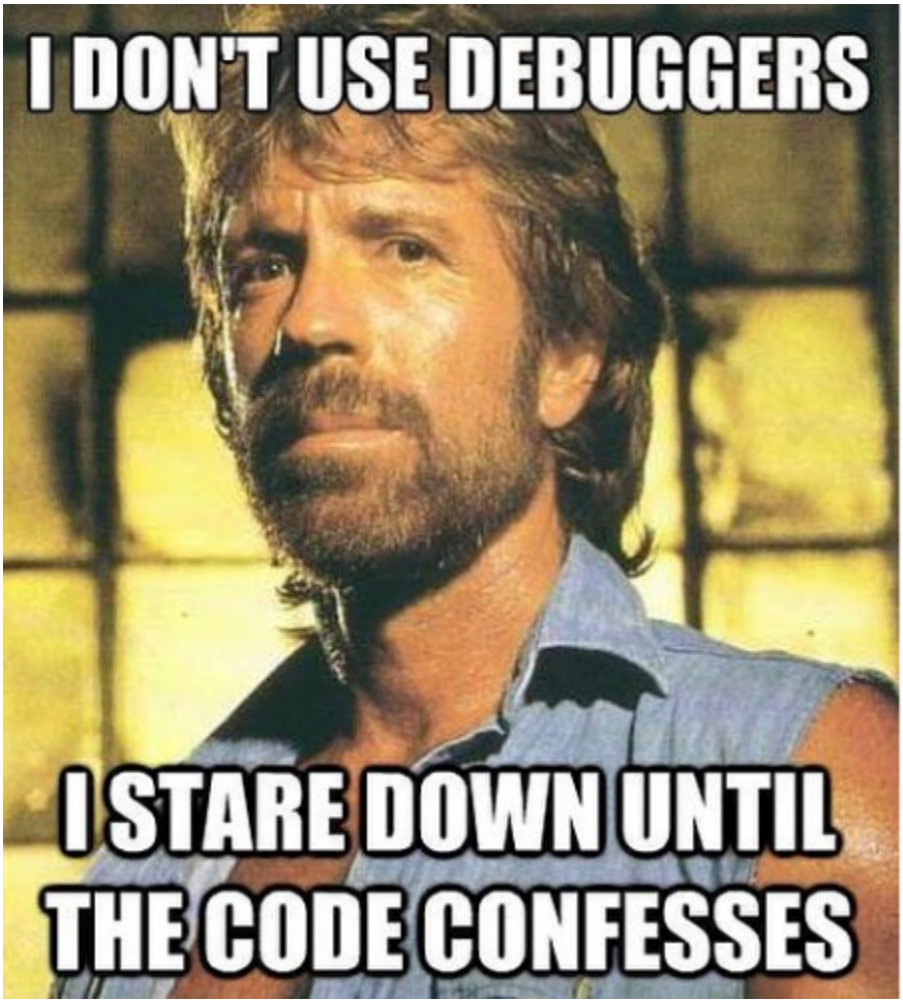
- Windows + WSL2 + Docker == Burnout.

- The kong and apisix took a lot of time for getting setup. Eventually, More of a pain was involved with the url on which the service was listening. Using host.docker.internal made the docker listen to services at Windows level and hence I had to expose the IP of WSL for docker to connect and work properly.

- Code without comments is painful, but a beautiful code is self explanatory. The below image shows my condition after checking the code I wrote last night.



- Debugging is not easy without debuggers, I just need to realise this.



Message for the Atlan Team

Hey Team,

It was really an amazing experience working on the problem statement. The task was new to me from every aspect of my coding experience. I actually learnt a lot about implementation of these microservices and system in the architecture. I'll really hope to have a next round of discussion on my architecture and discuss to make it more robust. Your feedbacks will surely be appreciated.

Thank You,
Deepankar


I.T. CONGRATULATING ME FOR
NOT FALLING FOR THE PHISHING TEST
ME WHO
DOESN'T
READ EMAILS